Intelligent Robotics - Team Four

http://flaws.redsdesk.de

Manuel Ebert maebert@uos.de
Maxime Louvel m.louvel@gmail.com
Vassilis Vassiliades vassilisvas@gmail.com

11 December 2007

Contents

---

1 Introduction

Intelligent robotics has became a widely studied field by researchers all over the world. Through this module we had the opportunity to peek into this field and find out for ourselves where the challenges are. For more than two months our team has been working on a rubbish collecting robot. We have all different backgrounds: Maxime studied embedded systems at the University of Lyon, France; Manuel studied cognitive science in Osnabrück, Germany and Vassilis just began his Masters degree in Intelligent Systems at the University of Birmingham. Each of us has brought his own knowledge and skills and we have tried to combine our backgrounds with the taught courses. Hence you will notice that our work has a certain focus on data analysis and machine learning techniques (you will also notice that unfortunately none of us had any background in engineering...).

1.1 Aims & ambitions

In the very first meeting that we had to discuss the principles of our robot we agreed that we wanted to take the chance to do things slightly different than the majority of teams throughout the last years did. One of the main things was that we did not want to follow walls to find the right corner but to have a vague representation of where the corners are relative to our robot, so that it could go straight to a corner without spending time on looking for it. This will be described in detail in *section 2.2*. The second most important thing was that we tried to predict the bottlenecks of the robot's performance and use more elaborate techniques there. We agreed that correctly classifying the different kinds of rubbish is one of the most crucial things; *section 2.4* describes how we pursued and compared different approaches to this task. Admittedly whilst performing extremely well on those "higher cognitive tasks" (for a LEGO robot, that is) the final robot showed certain deficiencies on the "lower cognitive" or motor tasks, which we will analyse and comment on in *section 3*.

1.2 Preliminary work

In order to get acquainted with hardware and software and the challenges connected with those we first built a robot that simply drives around the lab and avoids obstacles. We used a neural network to map sensor data to motor actions to produce emergent, local behaviour ([7]). One of the main advantages of using neural networks was that the robot was able to drive in very smooth trajectories to avoid walls and was able to drive around the lab without getting stuck on chairs or tables at all (however it did get stuck on obstacles as thick cables or uneven floor). *Figure 1* shows the final network.

The robot using this network is depicted in *figure 2*: It uses two IR sensors to detect obstacles and two bumpers that trigger micro switches to get the robot away from things the IR sensors missed. We could calculate some of the weights analytically ([10]), especially for the neurons with self-connection that will fire for a few of cycles after they have been initially excited, while other weights could only be tuned experimentally. The

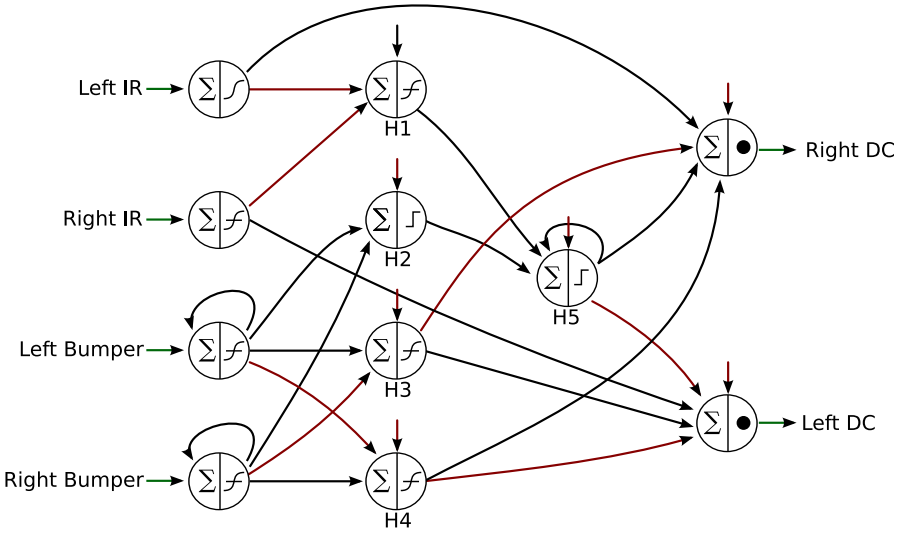


Figure 1: *The neural network used in the initial design. Red arrows are inhibitory synapses, black excitatory. H5 triggers a "reverse turn" behaviour if H1 (which fires if both IR sensors are below a threshold) or H2 (which fires if both bumpers were activated) excites it.*

beauty of this approach lies in the simplicity and the comprehensibility; however we decided against using the neural net again for the rubbish collecting robot as we ran into problems when integrating it with the other modules. One of the reasons for this was that we lacked the proper tools for changing, examining (on- and offline) and debugging large-scale networks on an embedded system, and how ever much we would have liked to implement and use these tools we probably would have not been able to deliver a robot that actually performed some task in time.

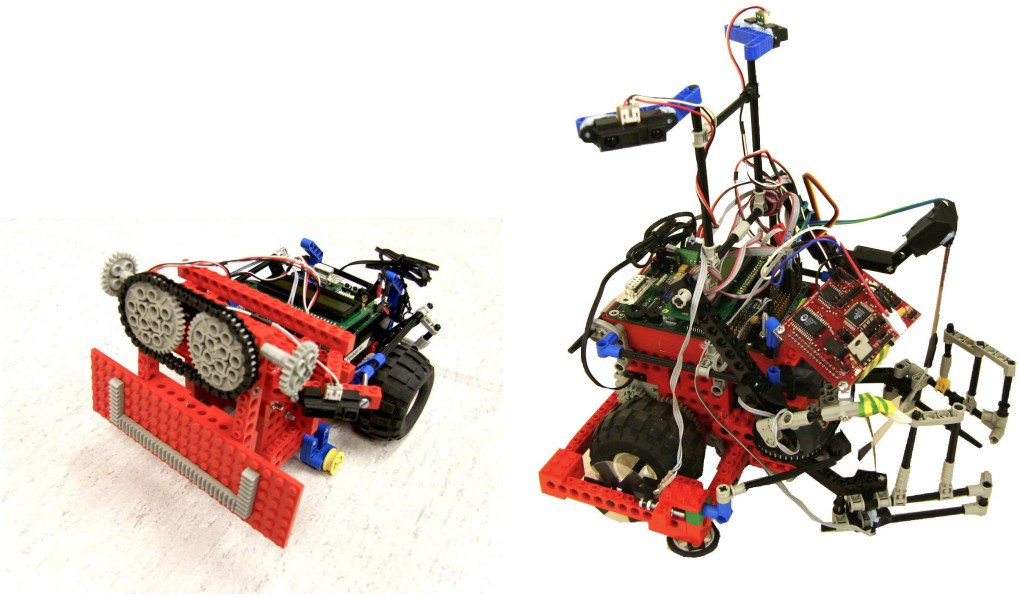
2 Robot Design

2.1 Mechanical Design

2.1.1 The initial robot

We will briefly present the design of the initial robot that used the neural network to drive around in the lab (cf. *figure 2 (a)*) as this revealed many of the challenges and difficulties in building a chassis that we tried to solve in the final version of the robot.

For the first robot we focused on simplicity and compactness provides as much agility as possible. Two large wheels were powered by two servo motors and a gear reduction of 16/40 (which allowed the robot to move really fast) with the IntelliBrain placed over the wheels to help the robot to turn smoothly. Everything else was built perfectly symmetrical: each side had one small wheel in the front to stabilise the robot, one IR sensor and a microswitch that was triggered by a bumper.



(a) (b)

Figure 2: *(a) The initial design. (b) The final design used in the competition.*

2.1.2 The final robot

The final task though was much more complicated that the initial one and since we knew that a lot of extra parts and sensors would be added, the whole robot had to be redesigned. Fortunately, the initial design showed some weaknesses of the chassis and the positioning of the sensors with respect to the final task, so by conducting some experiments we were able to overcome many limitations of the initial chassis. The three main improvements were done by limiting the weight of the robot by just using enough bricks to make a strong yet light chassis, replacing the two front wheels by a single one closer to the back wheels to make it easier to turn and finally reducing the gear ratio to 1/5. Before reducing the gear ratio we compared our new chassis (that now had to bear a lot more sensors) to the old chassis and could experimentally show that with the same motor power the new robot could move 40% faster and turn 35% faster than it did with the initial design, which means that we could either reduce the power of the motors (which the batteries would appreciate) or gear it down even further without much loss of speed.

Moreover, the short range IR sensors were replaced with medium range ones and placed higher than the height of the bottle used as rubbish in the final task. This is done mainly to avoid collision with walls. Following our minimalistic design ambitions we only used a single servo motor for sweeping the IR and opening and closing the arms.

We added two whisker sensors: one between the arms just above the lower IR sensor that is used to detect whether an object was grabbed and another whisker that was placed

higher than the height of the cans which helps in classifying the rubbish[1]. Unfortunately we discovered very late that the position of the lower whisker caused major problems, especially with tennis balls: because the balls were rolling forward when grabbed and pushed by the robot they sometimes took the whisker with them, what then caused the robot to think that it lost the object. Furthermore it sometimes happened that when scanning for rubbish the robot would lock an object between the lower whisker and the IR sensor, which on the one hand blocked the IR sensor and on the other hand prevented the robot from recognizing when it grabbed rubbish. This was in fact the major mistake we did with our design and if we had to do it again we would probably discard the idea of using only a single servo so that we could move arms and IR independently.

The classification of the other rubbish is done by using a CMUCam2 vision sensor (camera) which was placed at the same height as the higher whisker. However, the camera was not placed at the center of the robot, but slightly to the right, pointing directly towards the object so that it avoids eye-contact with the red blob on pepsi cans and can also be used for detecting corners. Additionally, a blue light emitting diode was placed below the camera as the natural blue channel of the camera is extremely poor. The final robot is shown in *figure 2 (b)*.

2.2 Localisation

This part deals with the localisation and the navigation of FLAWS . The navigation includes basic functionality as moving straight forward or reverse, turning θ radians to either direction and going to a corner. The localisation is to know where the robot is and where it faces.

2.2.1 Self-localisation

We quickly discarded the idea of building a detailed map as we thought that the task was perfectly solvable by only using local features of the environment. However we allow the robot to know roughly where it is at any time using the UMBmark method ([1]). The robot computes its position from its previous position and the movement it has done since the last known position. Our idea has been to correct the heading when the robot is lost, assuming it able to deduce its heading from its environment.

We have used the Intellibrain's API (`OdometricLocalizer`) to do the odometric computation. We used one breakbeam on each wheel to keep track of the revolutions of the wheels. The `AnalogShaftEncoder` class has been used to get the number of counts. This has worked quite well when we have done our first tests. However after integrating the individual components we got the heading completely wrong. We tried different things like change the period of the encoders and the localizer to finally until we finally came up with our own optimized implementation of the localizer. While we still run two threads for counting the number of turns made by each wheel, the computation of the x and y coordinates has been removed and the heading is now computed only when it is

[1] As described in *section 2.4*, we'd be able to classify the rubbish without the help of the upper whisker, however the whisker reduces the number of cases where the support vector machine has to be used.

needed directly from the encoder's values (as opposed to calculating the change of the heading in every cycle of the thread). The heading is now computed as follows:

$$heading = ((counts_{left} - counts_{right}) \cdot \delta_r \mod 2\pi) - \pi$$

with

$$\delta_r = \pi \cdot \frac{wheelDiameter \cdot countsPerRevolution}{trackWidth}$$

2.2.2 Navigation

There are three main methods of navigation used by the other parts of the robot:

- turn for a given angle
- go forward (or reverse)
- go to a corner.

After some frustrating experiments with using PID controllers for robot locomotion we decided that it is absolutely pointless to use PID controllers for turning to a specific angle or going straight. The main reason that led to this conclusion is the type of error signal that we have: in both cases the heading. As pointed out before the minimal measurable difference in our heading is δ_r, the radians the robot turns if it counts one transition of the break beams on one side. Hence the error signal is essentially non-continous. Let's now look at the control signal $M(t)$ of an PID controller with parameters K_p, K_i and K_d for the gains of the proportional, integral and derivational part at time point t,

$$M(t) = K_p e(t) + K_i \int_0^t e(\tau)\, \partial\tau + K_d \frac{\partial e}{\partial t}$$

As the error only changes in amounts of δ_r, the derivational part $\frac{\partial e}{\partial t}$ is either equal to 0 or equal to $|\delta_r|$ (assuming that the time steps are small enough so that every change in the break beams states happens in an individual time step. If not, things are even worse). Hence the control signal is boosted every time we measure a different heading, which inevitably causes overshooting and makes it impracticable to use the derivational part of the control signal. The integral part will be a non-continuous function with linear segments between the time point where a change of the measured heading occurs. The slope of these segments solely depends on the previous error and is simply $e(t_i)/\delta_r$. At this point keep in mind that the power that can be send to the motors is discrete as well and the motors need a certain minimal power to overcome the ground friction and inertia to turn (which can be different in different situations, this is why we wanted to use PID controllers in the first place). So if at some point t_i the integral part has increased enough to actually have an effect on the discretized motor power, the controller will necessarily have to overshoot before reaching a balanced state to make up for the

integral. If however the integral does not contribute towards the actual motor output it is unnecessary to compute it after all. Furthermore it is perfectly possible to stop the controller when the error is zero (although the controller is not yet in a balanced state) so that the robot will actually stop turning right where it is. Consequently all we need is the proportional part of the error signal. The next step is then to realize that instead of using the measured heading as an error signal we can avoid the calculation of the current heading by first calculating the number of shaft encoder transitions we need to correct our heading: $\delta_s = \delta_{heading}/\delta_r$. We can then not only distribute the rounded number of transitions that the left and the right encoder have to differ to turn for $\delta_{heading}$ radians to both motors but also remember the remainder of the rounding operation and add this to δ_s next time we call the method. This way it does not matter whether we turn 8 times $\pi/4$ radians or 2 times π radians as we accumulate our rounding errors:

$$\delta_s = \delta_{heading}/\delta_r + e_{remainder}$$

As said before we then split δ_s into one part for the left and one part for the right motor and map each part to a motor power (which in the end resembles a proportional controller). However, a few things can go wrong. Our problem is that the marks on the wheel used by the breakbeams are uniformly spread. One other problem could be if a break beam misses a transition, however this has almost never happens in our tests. Inaccuracies in the measurement's parameters that define δ_r are another minor source of errors.

Going to a corner: As mentioned before we have a vague representation of where the different corners are by the prior knowledge of their order and figuring out their absolute angle from the center of the arena. Given these values it is possible to move to a specific corner without having to go to any other corners in any case, as figure 3 illustrates.

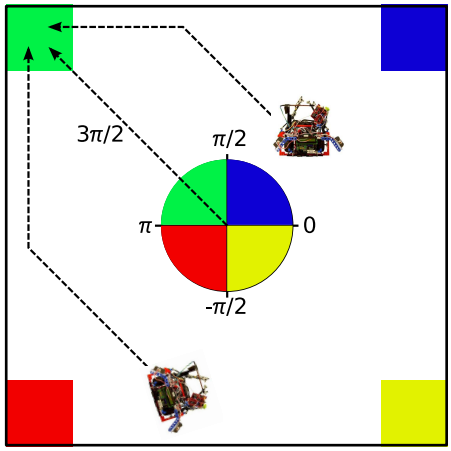


Figure 3: *Using heading information it is possible to go to a corner from anywhere in the arena with minimal detour.*

The procedure for doing so is as follows:

1. Turn to face the absolute angle specified corner (as if the robot was standing in the middle of the arena)
2. Go forward until detecting a wall with an IR sensor.
3. (If deemed necessary, face the wall to correct the heading)
4. Turn $\pi/2$ degrees (to the left if the right IR has detected a wall, otherwise to the right).
5. Follow the wall until reaching the corner (i.e. until the other IR sensor detects the opposite wall).

The bottleneck of this procedure is deciding whether to follow the wall on the left or on the right. This involves comparing the IR signals. We created individual models for each of the IR sensor to minimize the error by different biases, and thus the procedure can guarantee success if the error in the heading is below $\pi/4$, i.e. if we can infer whether we are left or right from the corner from the angle with which we arrive at the wall.

We used a PID controller for following the wall with the IR reading as the error signal as this signal can be easily smoothed and does thus not underlie the limitations discussed above. The target is to maintain a certain distance from the wall. However we had to constrain both the error and the control signal to produce sufficiently quick responses without overshooting. After tuning the PID controller using the Ziegler-Nichols method and some fine-grained manual adjustments ([17, 16]) the robot was able to follow the wall in all 22 tests if the initial distance of the respective IR sensor to the wall was within 12cm and 35cm.

The main way to enforce this condition is by first using the IR sensors to face the wall and then turn 90 degrees to the respective side. Doing so it can also correct the heading, as it will know which wall it is facing by the corner it meant to go to and the IR sensor that detected the wall first. If for some reason it confused the wall and thus got the heading wrong the "Check Integrity" behaviour, explained in *section 2.5*, will notice that the next time the robot reaches a corner.

Facing the wall is again done with a PID controller that minimizes the difference between the two sensor readings to 20cm.

2.3 Rubbish detection

2.3.1 Mapping measurements to actual distance

One phenomenon that most groups noticed at early stages is that the measures from the IR sensors heavily depend not only on the distance to the surface measured (as it should be) but also the material and the angle of the surface towards the sensor. We hence aimed at creating a mapping that will map a measurement of the Sharp GP2D12 IR sensor to the most likely actual distance. We will then test the model on real surfaces

and compare the results to the direct outputs of the IR class to show that our model reduces the error between measured and actual distance.

Method We first created a dataset using all four kinds of rubbish placed at 36 different distances ranging from 10cm to 80cm away from the sensor (which corresponds to the minimal and maximal ranges as stated by the manufacturer, cf. [6]) with five different angles. For every set up we took 20 samples with the IR sensor in order to gain further information about how many measurements are necessary so that the median of the measurements is robust to outliers.

We then aimed at finding a polynomial that fits the data, using only the median of a subset of the 20 measurements per set up and keeping the rest as a validation set. We first compared different sizes of subsets and came to the conclusion that if you select randomly select five out of the 20 samples, the median won't change if you add more samples in 95.27% of the cases, whilst this is only true for 83.47% of the cases if you first select four samples and then compare their median with larger subsets. We hence decided to take the median of five samples for all of our measurements.

The next task was to fit a polynomial to the data, which we could then invert to gain the actual model for the sensor. This was done by forming the Vandermonde matrix V such that $v_{i,j} = x_i^{n-j}$, where x_i is the ith measurement and n is the degree of the polynomial we want to find. For our data $n = 4$ seemed reasonable as the plotted data showed two "waves", hence we needed a function that allowed for at least two intersections of its second derivative with the x-axis (i.e. the sign of the curvature changes at least twice). Solving the system of linear equations $y = Vu$ will then give us the coefficients for a fourth-degree polynomial ([14, 11]). As this function is bijective within the domain of $[0, 80]$ we can find a definite inverse in this region, which is then our model for the sensor. In the case of the sensor used for detecting rubbish, the model was

$$y = -0.000015769\hat{x}^4 + 0.002589396\hat{x}^3 - 0.1362458\hat{x}^2 + 3.687646\hat{x}^1 - 19.29409\hat{x}^0$$

where $\hat{x}$ denotes the median of five measurements and y denotes the predicted distance to the object. *Figure 4* shows the polynomials for different kinds of rubbish. From this figure it is evident that tennis balls produce very outlandish sensor data, therefore disregarding the data from the tennis balls before fitting a polynomial to all data (regardless of the object that was measured) reduced the overall error on the training set.

In order to test our hypothesis that our model gives more accurate predictions of the distance than the raw sensor output, we first calculated the mean error and the standard deviation of the error between the actual distances and the medians of the five IR measurements for all objects and compared the values to the mean error and standard deviation of our model by performing T-Test.

Finally, to speed up the computation of the model, we stored a mapping of each actual distance with the estimated one from the function in a lookup table.

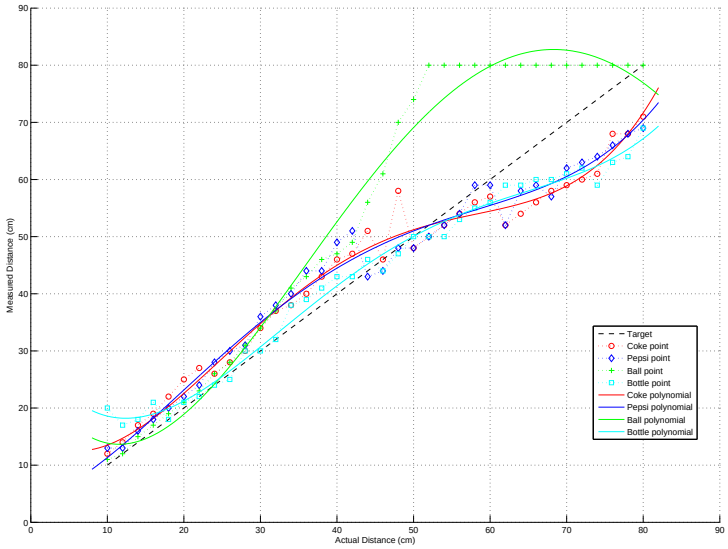


Figure 4: *Polynomials that approximate the medians of the measurements for each class of objects. Each point is the median of all measurements for that particular actual distance. The target line is the line* $y = x$.

Results & Conclusion Again we will exemplarily present the results for the model of the IR used for rubbish detection. Only using the medians of five measurements the mean error of all measurements was $\bar{e}_1 = 6.0563$ with $\sigma_1 = 5.6503$. For our function we got $\bar{e}_2 = 4.7746$ and $\sigma_2 = 3.0152$. For a paired T-test this yielded a P-value of 0.0096. The result indicates that the null hypothesis (that the errors were generated from the same random distribution) can be rejected at the 5% level. Based on these results, we can conclude that the robot is capable of estimating the distance to the objects more accurately when using this model. The estimation of the distance needs almost no computational power because the values are stored in a lookup table.

2.3.2 The detection algorithm

Our basic approach to detecting rubbish is to threshold the data by its median value minus γ standard deviations. The underlying assumption is that the median will account for the walls as in *figure 5 (a)* and we could use the gain parameter γ to adjust how sensitive the algorithm is towards rubbish. After thresholding the data we simply supposed that every peak that is left corresponds to one object.

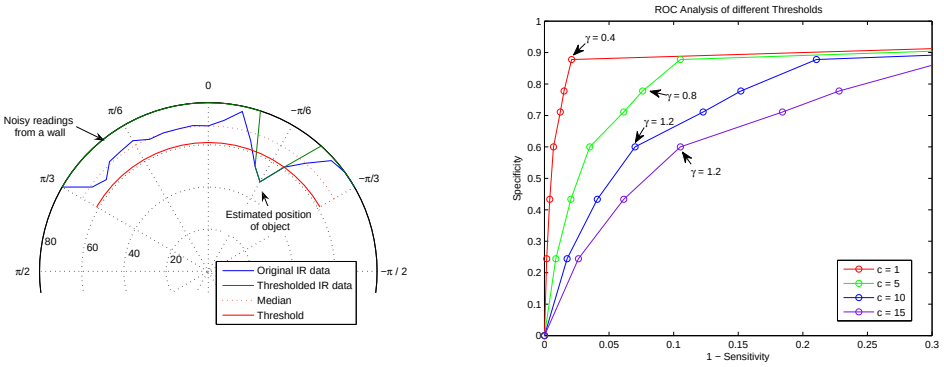
We evaluated the algorithm (as implemented in MatLab) on a dataset with 60 samples, containing a total of 90 objects. A object was considered to be correctly identified if its estimated angle is not more than one servo step away from its actual angle towards the robot (although this id admittedly vague as the objects had different distances to-

wards the robot). Every peak that did not correspond to any object counts towards the false positives. As we allow for a misplacement of one servo angle the actual maximal number of positives that could be found is a third of the number of servo steps, in our case $\frac{21}{3} = 7$. *Table 1* shows the results of the evaluation using 5-fold cross validation.

γ	TP	FP
0.4	0.78	0.0287
0.6	0.87	0.0211
0.8	0.77	0.0152
1.0	0.71	0.0123
1.2	0.60	0.0070
1.4	0.43	0.0041
1.6	0.24	0.0018

Table 1: *The ratio of true and positives and false negatives for varying γ. Note that the results for $\gamma = 0.6$ are never optimal as this point in FP-TP-space is within the convex hull of the other points.*

The cost c of the false positives depends on how many objects are left in the arena: If there are many objects, than false positives have a high cost, i.e. trying to grab something that does not even exist takes far more time than doing a detour because the robot did not recognize something. This cost has to decrease with every object that we successfully remove. We analysed the results for costs of 1, 5, 10 and 15, as depicted in *figure 5 (b)*.



(a) (b)

Figure 5: *(a) The algorithm correctly identifies the object at $\pi/6$ radians at a distance of 50cm against the wall at a distance of 70cm. (b) The ROC curve varying parameters c and γ. Non-optimal results (under the convex hull) are omitted.*

Hence the optimal threshold parameter γ^* can be selected as follows, where $sensitivity_\gamma$ and $specificity_\gamma$ denote the specificity and sensitivity of the algorithm for a given γ and

c denotes the cost of false positives:

$$\gamma^* = \arg\max_{\gamma} \sqrt{(1 - sensitivity_\gamma)^2 + c \cdot specificity_\gamma^2}$$

2.4 Classification of Objects

Our robot has to solve three primary classification tasks:

1. Quickly classify rubbish versus non-rubbish
2. Classify different types of rubbish
3. Classify the colour of corners

As described in *section 2*, we use two whiskers and the CMUCam2 for acquiring the data necessary for classification. From the CMUCam2 we solely use the statistical data (mean values and standard deviations for the three colour channels). For the second task we gathered 200 data samples with objects from different angles and with different light conditions to create a training set and another subset without the standard deviations. The data set for the third task is comprised of 20 samples from each corner. The actual camera data only contained the difference to the initial calibration of the camera, which made it even more robust towards changes in light condition and has then been normalized (which is only important for support vector classifiers, see [13]). We then trained support vector machines, decision trees and logistic model trees on the different data sets. For the first task we simply chose a threshold θ^* that minimizes the entropy of the two subsets it creates (cf. [9]):

$$\theta^* = \arg\min_{\theta} -p(rubbish|x \geq \theta) \log_2 p(rubbish|x \geq \theta) - \\ (1 - p(rubbish|x \geq \theta)) \log_2 (1 - p(rubbish|x \geq \theta))$$

where $p(rubbish|x \geq \theta)$ denotes the probability that we have touched an object with the whisker if the value sampled by the whisker is greater or equal θ. Moreover our earlier experiments have shown that for none of the algorithms there is a significant improvement if the standard deviations of the colour channels are given, hence we will now focus on the latter two tasks with the reduced training set to find a suitable algorithm.

2.4.1 Support vector machines

Support vector machines (SVMs) have received great attention over the past years since their introduction in 1992 ([2]). The aim is to maximize the margin

$$\rho = \min_i \left| \frac{\langle \vec{x}^{(i)}, \vec{w} \rangle + w_0}{||\vec{w}||} \right|$$

from a pattern to the decision boundary. This can be rewritten as an actual learning task (with class labels $d \in \{-1, +1\}$, slack variables ξ, a cost parameter C and a kernel function K_Φ):

$$\underset{\vec{w},w_0}{\text{minimize}} \frac{1}{2}||\vec{w}||^2 + C \cdot \sum_{i=1}^{p} \xi_i^2$$

$$s.t. \quad d^{(i)}(K_\Phi(\vec{x}^{(i)}, \vec{w}) + w_0) \geq 1 \quad \text{for } i = 1...p$$

which can then be solved through maximizing the Lagrange coefficients $\alpha_1...\alpha_p$ (which will be zero for all patterns that don't contribute towards the decision boundary) of the Lagrange function $L(\vec{x}, \vec{\alpha})$ by solving the dual problem

$$\underset{\vec{\alpha}}{\text{maximize}}\, Q(\vec{x}, \vec{\alpha}) = \underset{\vec{\alpha}}{\text{maximize}} \inf_{\vec{x} \in \mathbb{R}^n} L(\vec{x}, \vec{\alpha})$$

The kernel function $K_\Phi(\vec{x}, \vec{y}) = \langle \Phi(\vec{x}), \Phi(\vec{y}) \rangle$ allows for a non-linear mapping to a higher (or even infinitely) dimensional space and the slack variable ξ is the error of each pattern, thus allowing "soft margin" classification for tasks that are (in feature space) not linearly separable.

For training our models we used *libsvm* ([5]). We created a heavily reduced version of the prediction algorithm optimized for the limited powers of the IntelliBrain which we made available through our team's website (`http://flaws.redsdesk.de`).

Classifying rubbish For the second task we performed a grid search in parameter space to find suitable values for C and the γ coefficient for the radial basis kernel function $K_\Phi(\vec{x}, \vec{y}) = e^{(-\gamma \cdot |\vec{x} - \vec{y}|^2)}$ and evaluated different parameters with 10-fold cross-validation. The best results were obtained with $C = 126$ and $\gamma = 3.2$, obtaining an overall accuracy of 87.05% with a standard deviation of 0.26 after 10-fold cross-validation. Detailed results are listed in *table 2 (a)*.

Classifying corners If the cost parameter C is high enough (roughly $C \geq 40$) the SVM will classify all patterns correctly in all of the folds. Detailed results and comparisons for this tasks can be found in *table 2 (b)*.

2.4.2 Decision trees

Decision tree algorithms ([3]) are a fast and simple way of learning boundaries between classes. We used an algorithm based on C4.5 ([12]) with post pruning to avoid overfitting and best first search based on information gain analysis to find suitable decision values. As the algorithms are commonly known we omit further explanation at this point.

Classifying rubbish After pruning the tree, 41 decision nodes were left, and the accuracy after 10-fold cross-validation was 71.19% with a standard deviation of 0.33.

Classifying corners For the third task, the algorithm learned a model with eleven nodes and classified 91% of the samples correctly, with a standard deviation of 0.18 over the 10 folds used for cross-validaton.

2.4.3 Logistic model trees

LMTs are decision trees with logistic regression functions as leaves, which overcomes the limitation of simple decision trees that boundaries can only be perpendicular to the axes. Again, details of the algorithm are described in [8] and [15].

Classifying rubbish The LMT algorithm learned a tree with eleven nodes and six leaves. For each of the leaves it calculated a linear function to separate different classes. In this manner it achieved an total accuracy of 79.74% with an standard deviation of 0.28 after 10-fold cross-calidation.

Classifying corners In this case the LMT simply learned one linear boundary for each class, i.e. the decision tree contained only a single node. It classified all but one pattern correctly in most folds (99% accuracy, standard deviation of 0.061).

2.4.4 Evaluation & Conclusion

Table 2 shows the results for the individual classes and the average time that it takes to evaluate the models on the IntelliBrain (with the standard deviations of ten measurements being always below 30ms). For the second task it is immediately obvious that the ROC areas of the SVMs are always equal or greater than the competing algorithm's areas while there is no significant difference between SVMs and LMTs in the third case. However predicting new instances takes much longer time – most of the time goes into evaluating the RBF kernel for the instance and each of the 59 and respectively 22 support vectors. Hence for the third task the logistic model tree is definitely the better choice.

(a)

	DT		LMT		SVM	
	TP	FP	TP	FP	TP	FP
Coke	0.62	0.17	0.79	0.16	0.82	0.06
Bottle	0.92	0.01	0.95	0.01	0.95	0.01
Pepsi	0.58	0.12	0.68	0.08	0.82	0.08
Ball	0.71	0.08	0.76	0.02	0.89	0.02
Total	0.711		0.797		0.870	
Time	272		424		6173	

(b)

	DT		LMT		SVM	
	TP	FP	TP	FP	TP	FP
Floor	0.95	0.01	1.00	0.00	1.00	0.00
Red	1.00	0.02	1.00	0.00	1.00	0.00
Green	0.95	0.06	1.00	0.01	1.00	0.00
Blue	0.75	0.01	0.95	0.00	1.00	0.00
Yellow	0.90	0.00	1.00	0.00	1.00	0.00
Total	0.91		0.99		1.0	
Time	272		424		6173	

Table 2: *The true and false positive rates of (a) the second and (b) the third classification task and the time it takes to evaluate the model on the Intellibrain (in ms).*

To overcome the high computational effort for the second task we combined the LMT and the support vector model: our classification algorithm will first evaluate the logistic model tree and calculate the distance of the pattern to the relevant decision

boundaries. Only this distance is below a threshold τ it will use the SVM to predict a class for the model. Since the misclassified patterns of the SVM were a strict subset of the misclassified patterns of the LMT (which of course is not generally true) we were able to find a τ such that our combined model produces exactly the same results as the support vector classifier, however for 141 out of 200 samples the computationally cheaper logistic models can be used for classification. Consequently the average time it takes to predict the class of an object is about 2120ms under the assumption that our dataset resembles the actual distribution of patterns in the input space. In fact we think that our dataset contains far more patterns that are close to the decision boundary (thus being "weird" cases) than we will find in the arena as the SVM was rarely really used.

2.5 The main control cycle

When it comes to the top-level behaviour, most rubbish collecting robots will probably follow the same basic loop: *Look for rubbish → grab rubbish → classify it and take it to the respective corner → look for rubbish* etc. Whilst for the one-object-at-a-time design of the robots this is arguably the best strategy, it does not define behaviours for non-default situations, e.g. when the object is lost on the way to a corner. Thus we chose a mildly behaviour-based architecture in the style of Brooks ([4]) with sub-behaviours that perform the actual actions so that the dominant behaviour depends on situation dependent variables rather than the current position in the program code. Apart from the addition of sub-behaviours we rather call it "mildly" behaviour-based as it is not entirely reactive, e.g. sophisticated algorithms are used for object classification and interpretation of IR data and a (very vague) representation of the non-local environment. The main mutually exclusive behaviours are depicted in *figure 6*. In the following subsections we will briefly discuss the key concepts of each behavior.

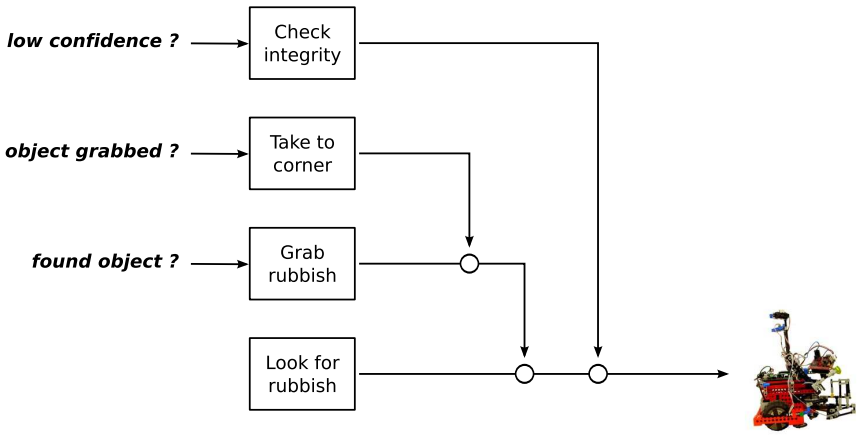


Figure 6: *The top-level behaviours comprise of many sub-behaviors that only compete within the currently active top-level behaviour.*

2.5.1 Looking for rubbish

The functionality of the analysis of the data the sweeping IR gathers has already been discussed in *section 2.3*. The question is now where and when to sample and analyse data. The main variable here is an internal frustration level that increases over time if no objects have been detected. If the frustration is low, the robot will examine his immediate surroundings first by turning left and right and moving a little (cf. *figure 7*). The more frustrated it becomes the more willing it will be to take long drives through the arena to look for rubbish, and also the more fine-grained it will scan its surroundings (i.e. the smaller the angle between two scans will be). As said before we did never want to rely on precise mapping, however our original idea was to cluster the arena into different areas and remember how many objects we have "seen" in each area to avoid long walks when there are few objects left in the arena – this was meant to loosely resemble a humans intuition of which direction to go to without explicitly stating where exactly in space these objects are. Unfortunately this idea fell victim to time spent on less interesting issues, hence frustration and doggedness are the only concepts used in this behaviour.

2.5.2 Grabbing objects

After turning towards the angle where an object is assumed the robot will open its arms and drive towards the object – or half way if the estimated distance is greater than 35cm in which case the behaviour's demand control will expire in favour of the rubbish searching behaviour. Knowing that turning the robot has an mean error of $e_t = \pm 0.0455$ radians and going straight for results in a mean error of $e_f = \pm 0.06$ radians per meter (as discussed in *section 2.2.1* and *2.2.2*), the angle between two servo steps is $\delta_s = 0.017$ radians (assuming that our algorithm identifies the right sensor reading as being caused by an object), we can estimate that when attempting to grab an object in r cm distance the mean error in our horizontal position will be lower[2] than

$$e_{grab} \leq (e_t + e_f \cdot r + \delta_r) \cdot r$$

The diameter of our arms (when opened in a way that we can detect collision with objects with the whiskers) is $14cm$, and the diameter of a can is $6.5cm$ so we should not go farther than some r such that $e_{grab} \leq \frac{14-6.5}{2}cm$. From the equation above we can deduce that

$$(e_t + e_f \cdot r + \delta_r) \cdot r \leq 0.0325$$

$$r \leq 0.3808$$

Hence, $35cm$ seems like a good threshold after which to stop and scan again. If the robot drives too far it will realize this and reverse before giving control to the rubbish detector again.

[2] In a happy robot world the different errors would cancel each other out.

2.5.3 Going to a corner

The basic functionality of this behavior has been discussed in *section 2.2.2*, so at this point we shall only note that due to the behaviour based architecture the procedures described in the before mentioned section can be at any time interrupted if other behaviours seek control, the robot lost its rubbish or is not sure where to go to anymore.

3 Evaluation & Conclusion

Figure 7 depicts three typical runs of the robot over five minutes. The brown stems correspond to the problem with the lower whisker reported in *section 2.1.2*.

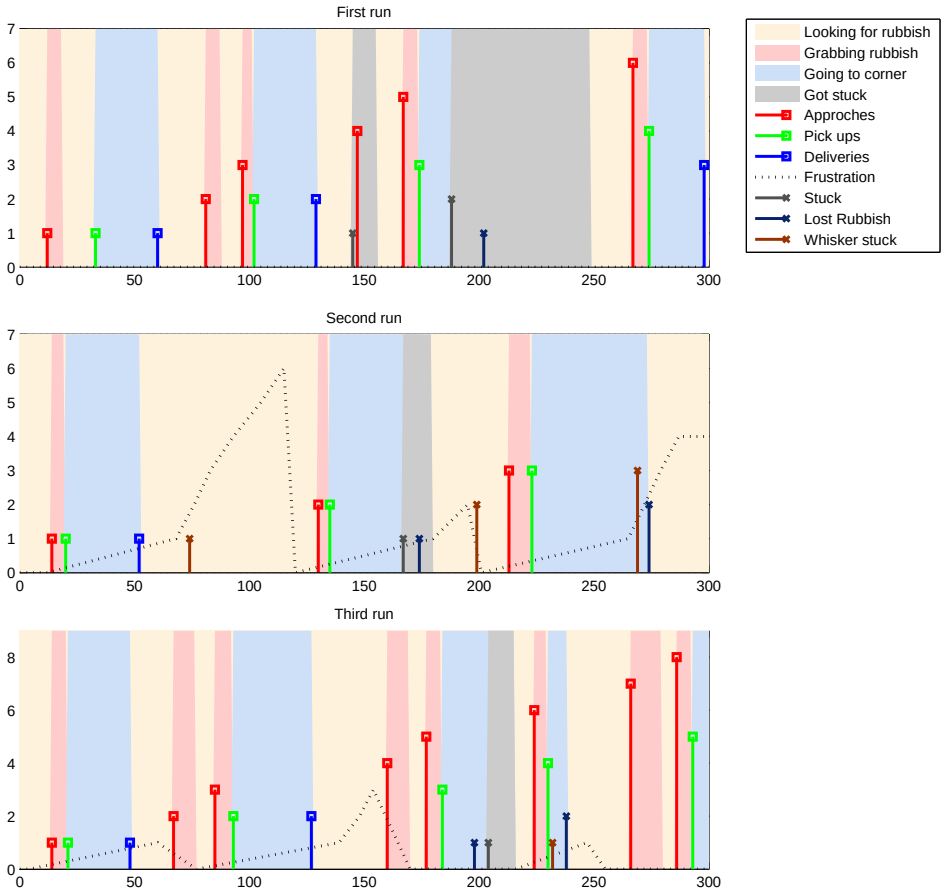


Figure 7: *A visualization of three runs of five minutes (300 seconds on the x-axis) each: the shading of the background shows which behaviour was dominant over time (or whether the robot got stuck), the stems show when how often certain events happened.*

One of the obvious problems is that the robot tends to get stuck on walls quite often. Although is it usually capable of releasing himself through timeouts and reversing

behaviours getting stuck has a far more severe impact on our robot as it would have on other robots since it easily looses track of its heading if a wheel turns without ground contact. Therefore human intervention is almost always required if the robot got stuck. The two ways around this would be to either use a particle filter or to disregard the heading and use simple wall-following algorithms as the other teams did. Whilst the first option would certainly be the more interesting one the latter might have been the more promising approach. However we did not want to abandon (what we believe to be) good ideas because other ideas are just easier to implement. Considering the tremendous amount of time spent on the odometry we think that we could have achieved far better results if we would have been able to use a compass for determining the heading and thus spend more time on the real "intelligence" of the system rather than becoming more and more frustrated on the supposedly simple task of turning the wheels with the right speed (after all the module is called "*Intelligent* Robotics").

A comment on the embarrassing performance during the demos: as mentioned in the introduction, the robot performs well on classification tasks, however the basic movement did not work incredibly well. We believe that most of the misbehaviour was caused by two parameters: the minimal and maximal angle of the IR sensor that was connected to the servo by two gears. We noticed that in our test after the demos the robot always drove past the object it wanted to grab, and always on the right side. We assume that in the night before, when working on other minor tasks, we accidentally turned one of the gears causing the IR to move to different positions for the same servo position. This in turn messed up the translation between the servo positions where we found objects and the angle that the robot should turn, causing the robot to turn too far to the right and miss the objects each time.

However there is no excuse for getting stuck at walls and we admit that this is due to mistakes made in the design phase.

3.1 Summary

In this report we presented the ideas and considerations behind the design of our robot and the algorithms it uses. We believe that we have pursued several novel approaches, among them using neural networks for controlling the robot, representation of the position of corners and state-of-the-art classification of rubbish. Much of the time that we spent on this robot went into research, implementation and testing of these methods, and while some have proved very useful others did not perform well or didn't make it into our final design at all. We regret that our robot's overall performance had to suffer from the focus on relatively sophisticated yet "smaller" modules, however we think that this was well worth the effort as all of us learnt a great deal about using such techniques on actual robots, and by providing the SVM libraries[3] for the IntelliBrain for the upcoming "Intelligent Robotics" modules we hope that future students will be able to use some of our work to spend more time on the actual "intelligence" of the robot.

[3]Available through our team's website, `http://flaws.redsdesk.de`

References

[1] J. Borenstein and L. Feng. UMBMark - a method for measuring, comparing, and correcting dead-reckoning errors in mobile robots. December 1994.

[2] B. E. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, 1992.

[3] L. Brieman, J.H. Friedman, R.A. Olshen, and C.J. Stone. Classification and Regression Trees. *Wadsworth Inc*, 67, 1984.

[4] Rodney A Brooks. A robust layered control system for a mobile robot. *IEEE Journal Of Robotics And Automation*, 1986.

[5] C. Chang and C. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[6] Sharp Electronics. Sharp gp2d12 datasheet. Retrieved from `http://www.acroname.com/robotics/parts/SharpGP2D12-15.pdf`.

[7] D Janglová. Neural networks in mobile robot motion. *Inernational Journal of Advanced Robotic Systems*, 1(1):pp 15–22, 2004.

[8] N. Landwehr, M. Hall, and E. Frank. Logistic Model Trees. *Machine Learning: Ecml 2003: 14th European Conference on Machine Learning, Cavtat-Dubrovnik, Croatia, September 22-26, 2003: Proceedings*, 2003.

[9] D.J.C. MacKay. Information theory, inference, and learning algorithms. 2003.

[10] Frank Pasemann. Dynamics of a single model neuron. *International Journal of Bifurcation and Chaos [in Applied Sciences and Engineering]*, 3(2):271–278, 1993.

[11] MJD Powell. *Approximation Theory and Methods*. Cambridge University Press, 1981.

[12] J.R. Quinlan. *C4. 5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[13] W.S. Sarle. Neural Network FAQ, part 1 of 7: Introduction, periodic posting to the Usenet newsgroup comp. ai. neural-nets. *URL: ftp://ftp. sas. com/pub/neural/FAQ. html*, 1997.

[14] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis*. Springer, 2002.

[15] M. Sumner, E. Frank, and M. Hall. Speeding Up Logistic Model Tree Induction. *Lecture notes in computer science*, pages 675–683.

[16] V. VanDoren. Loop Tuning Fundamentals. *Control Engineering*, pages 30–32, 2003.

[17] J G Ziegler and N B Nichols. Optimum settings for automatic controllers. *Journal of dynamic systems, measurement, and control*, 115(2 B):220–222, 1993.